Instituts thématiques

Inserm

Institut national
de la santé et de la recherche médicale

IBM Software Group

Formal methods integration in the software development process

Isabelle Perseil (INSERM), Philippe Leblanc (IBM)

- -1- Introduction
- -2- A methodological context
- -3- RUP advantages / shortcomings
- -4- Real-time languages and best practices
- -5- Formal methods integration in real-time software development
- -6- From RUP to the C-Method software development process
- -7- Conclusions

- The main tool of project management
- An integrated process in a methodological approach
- State-of-the-art of good practices in software process development techniques for DRES
- RUP and DRES
- Discrepancy between
  - Evolution of modeling languages, practices of model transformation and verification AND
  - Evolution of the processes which use them during the phases of requirements specification, analysis, design and certified code generation
- Very rare integration approaches in industrial environments
  - Method B at RATP
  - Esterel at Dassault
  - Intensive use of PVS at NASA
- Our approach
  - To enrich the current process with other phases
  - Consider that the requirements of strategic type must first be completely identified, specified, verified
  - Parallelization of sub-processes
  - A seamless development involving intermediate languages

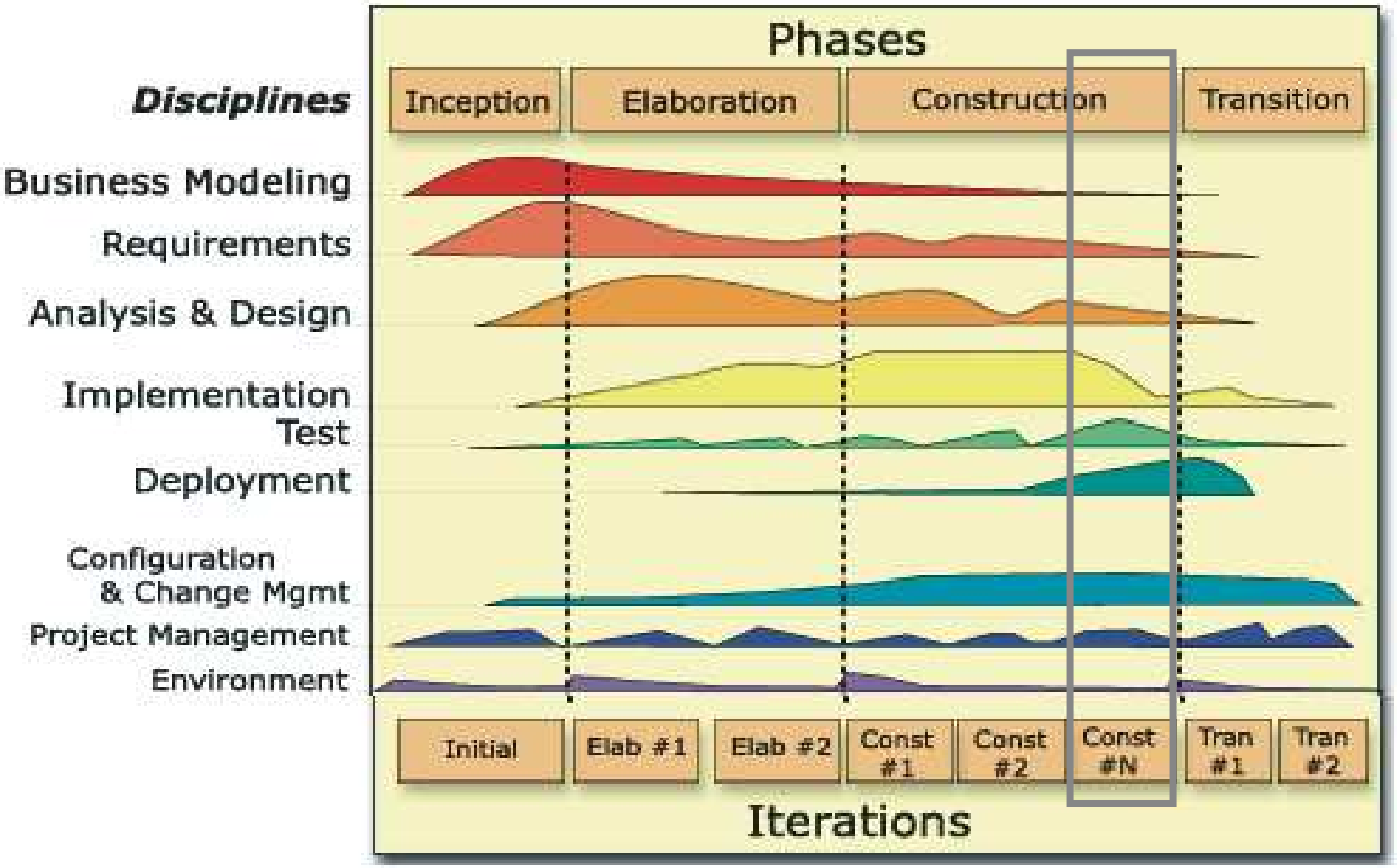
- A software development process can be applied with many methodologies or many methods (or without any)

- The concept of software development process is closely related to the concept of task scheduling, and it is essentially a matter of temporal order
  - The underlying field is **project management**.
  - How do we manage avionic embedded systems projects?
    - Respect of the standards (DO178B and ED-12-B)
    - Modularity
    - Check lists

2/ A process within a method

- A method is built from a particular methodology and is adopting only one software development process
- The concept of method is closely related to the concept of strategy.
- The method provides means for executing the development process tasks in an optimal way
  - The method refines the methodology with a set of improvements.
  - The underlying field is **Operations Research**.
- This concept goes beyond the development process concept
  - there is not a unique way of performing an activity, whatever is the issue
- At the level of a method, any type of issue has to be considered, with the optimal way of solving it
  - there is a logic that allows us to solve any issue with more efficiency
  - the method shows in details ***how to apply this logic***

1 / RUP iterations



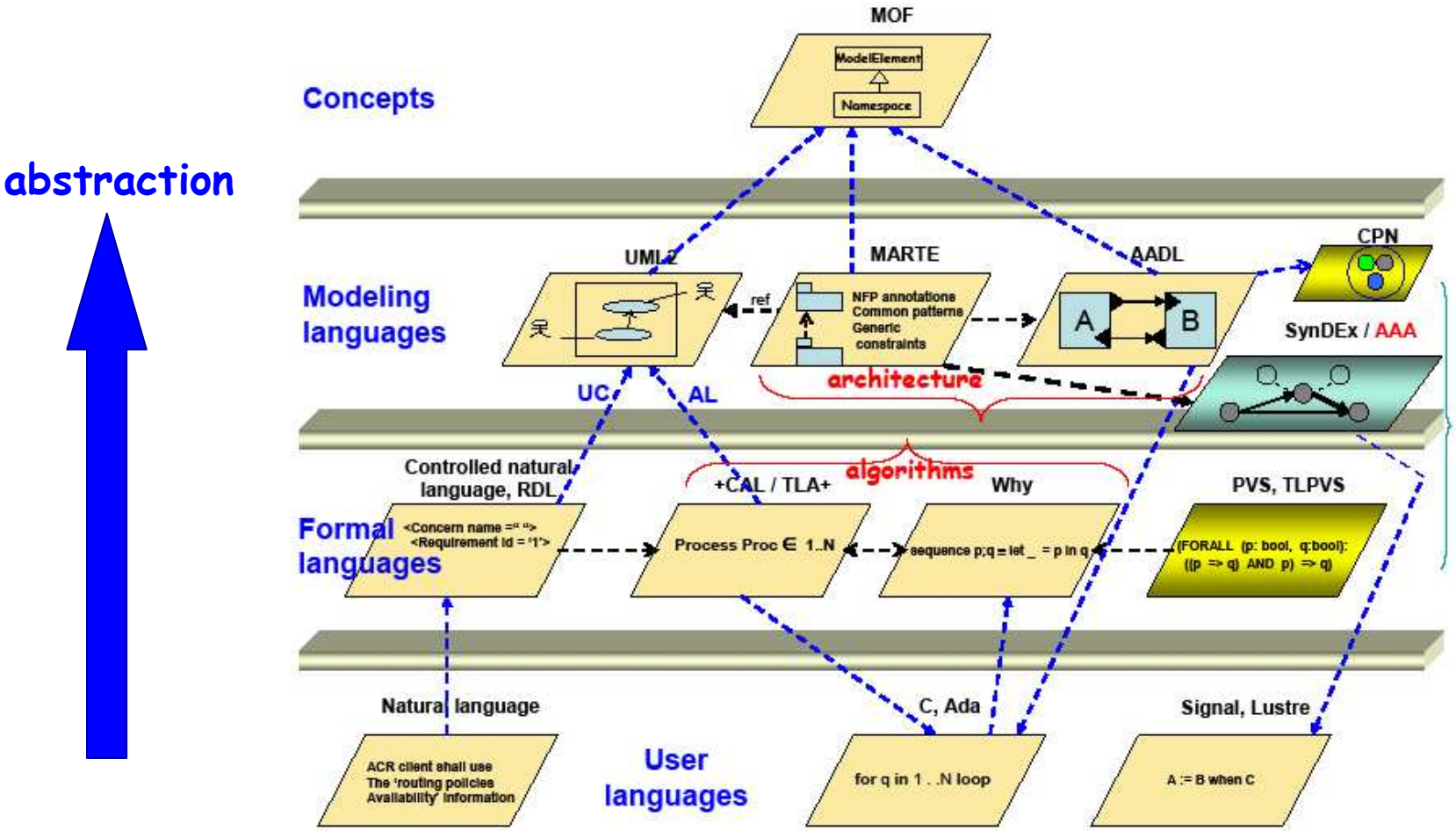
2/ RUP advantages

- IBM Rational definitely banished the waterfall process
  - Unfortunately this good resolution have not been followed by everyone, even in the research field.
- The ``use case driven'' approach is definitively a very good approach that is even kept in the Agile methods
  - allows the requirements to be traced
- The ``architecture-centric'' process is adopted for all complex and large systems
- The possible customization enables an adaptable process framework in which each company may choose the most convenient elements.

3/ RUP shortcomings

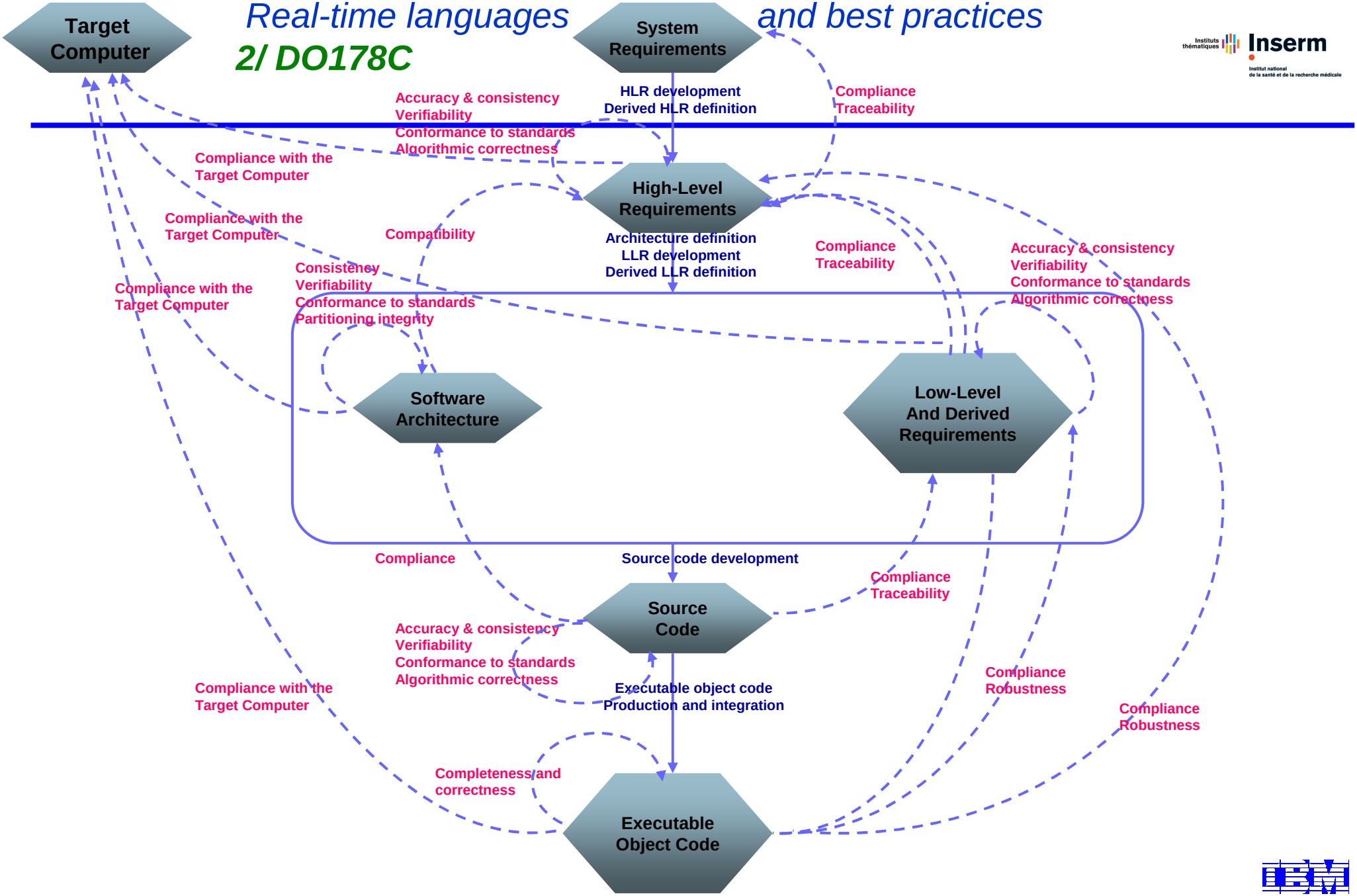
- The homogeneous decomposition between Inception Elaboration and Construction is too much simplistic
  - Because depending on activities types, cycles are more or less complex, therefore not homogeneous
- The RUP is supported by a very heavy tool, which is not intuitive
  - The learning period is long and requires significant investments
- Depending on the environment, the parameterization may also be very long
  - the parameterization gives the impression of genericity,
  - but the process is not fundamentally different for a any kind of project (telecommunications, automotive, aeronautics, financial, etc) : the phases and activities are the very same.
- The RUP is only suitable for very big projects
  - its intrinsic logic is so much linked to the IBM Rational world that it is mostly applied with the entire tool suite.
- The entire process is rather a set of good recipes than the result of a rigorous ``rationale'' as the name should suggest
  - The inception phase should be global with respect to a systemic approach.

1/ Languages and their abstraction levels

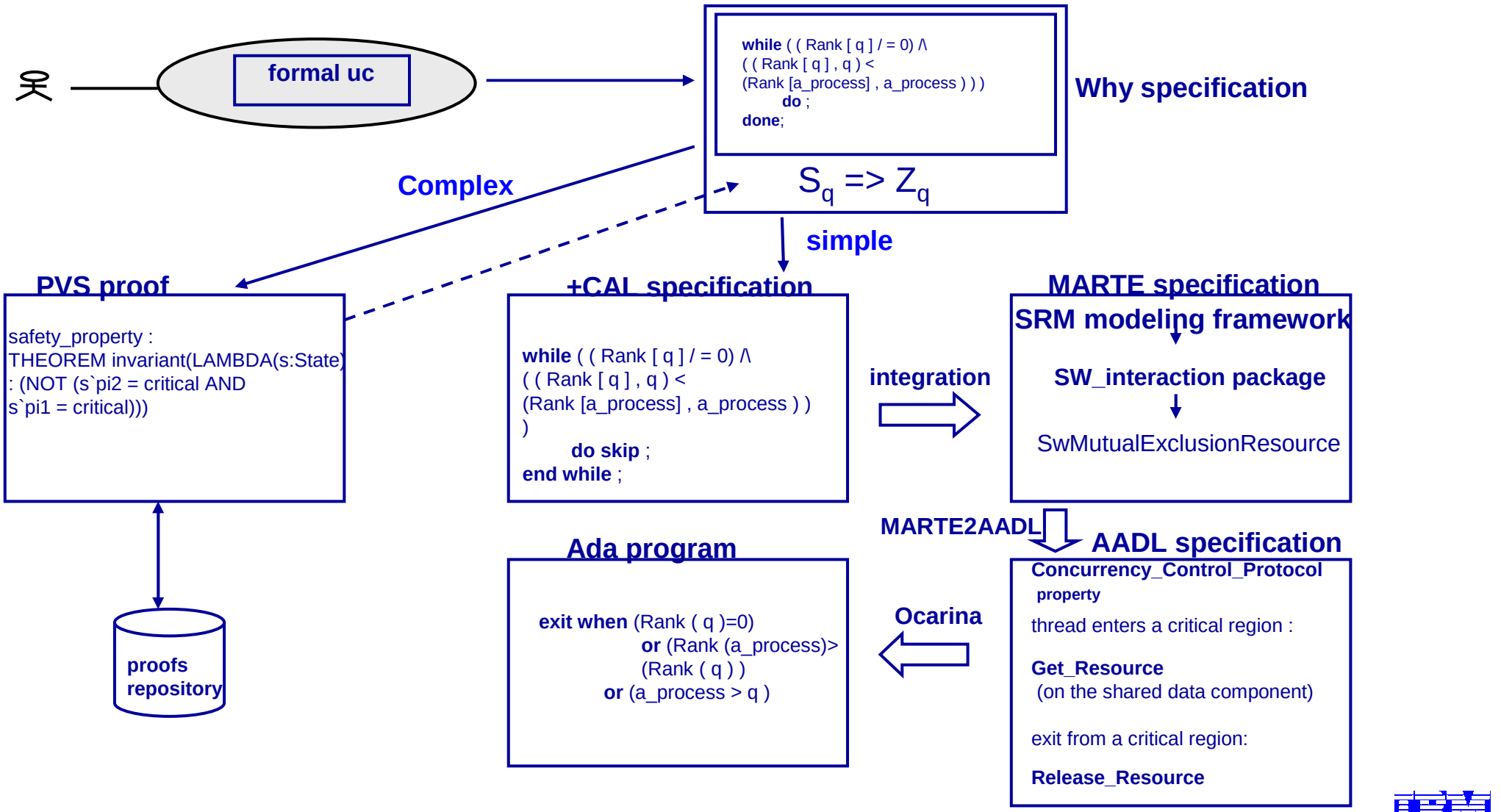


Real-time languages and best practices

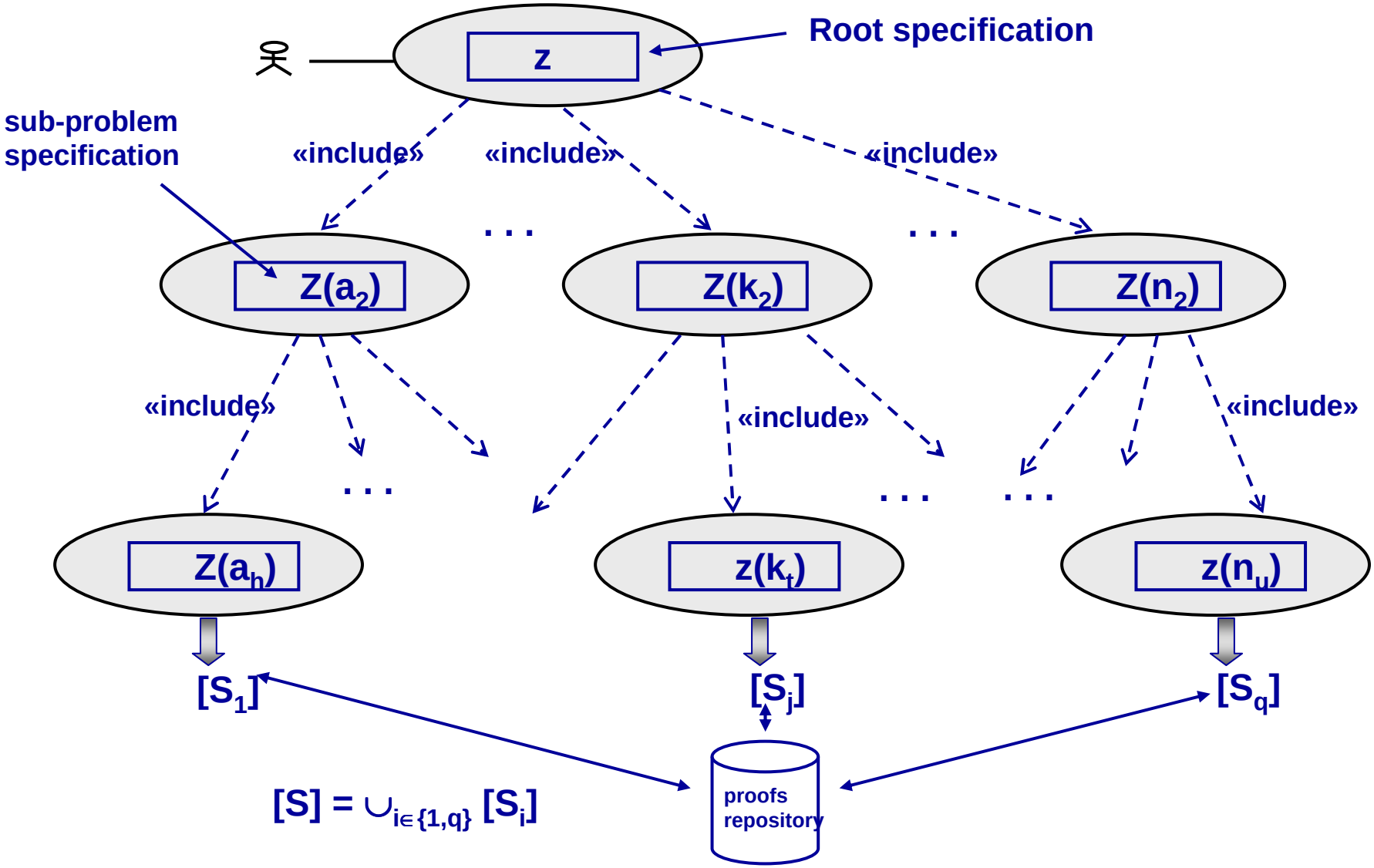
2/ DO178C



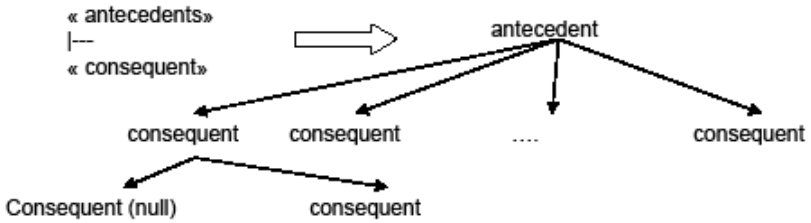
1/ a formal use-case driven method



2/ Proof-based use cases: a sub-objectives technique



3/ Proof-based use cases: a sub-objectives technique



$$Z \vdash S$$

$$Z_1 \vdash S_1$$

$$Z_{11} \vdash S_{11} \qquad Z_{12} \vdash S_{12} \qquad Z_{13} \vdash S_{13}$$

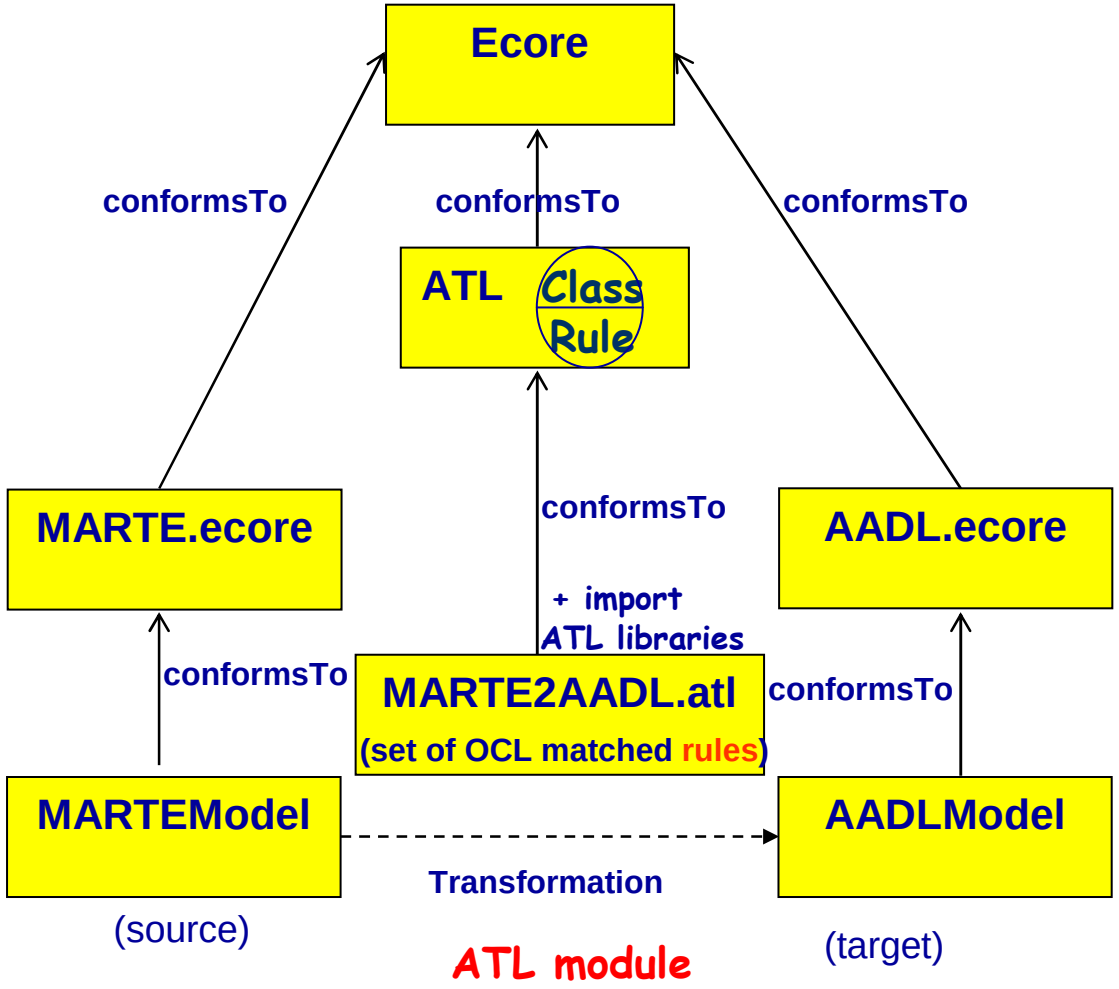
4/ Architecture Model transformation: MARTE subset to AADL with ATL

… Corresponds to …

- <<swSchedulableRessource>> → Thread
- → Thread group
- → Shared Data…
- → Subprogram…
- → Processor
- → Memory
- → Device…
- → System
- → Port…
- → Mode
- …
- → System Binding
- …

rules established from source model to target :
Stereotype MARTE to AADL element
Ex:
<<swSchedulableRessource>> to Thread

OCL :
do {thismodule.dsl(t, Thread);
}

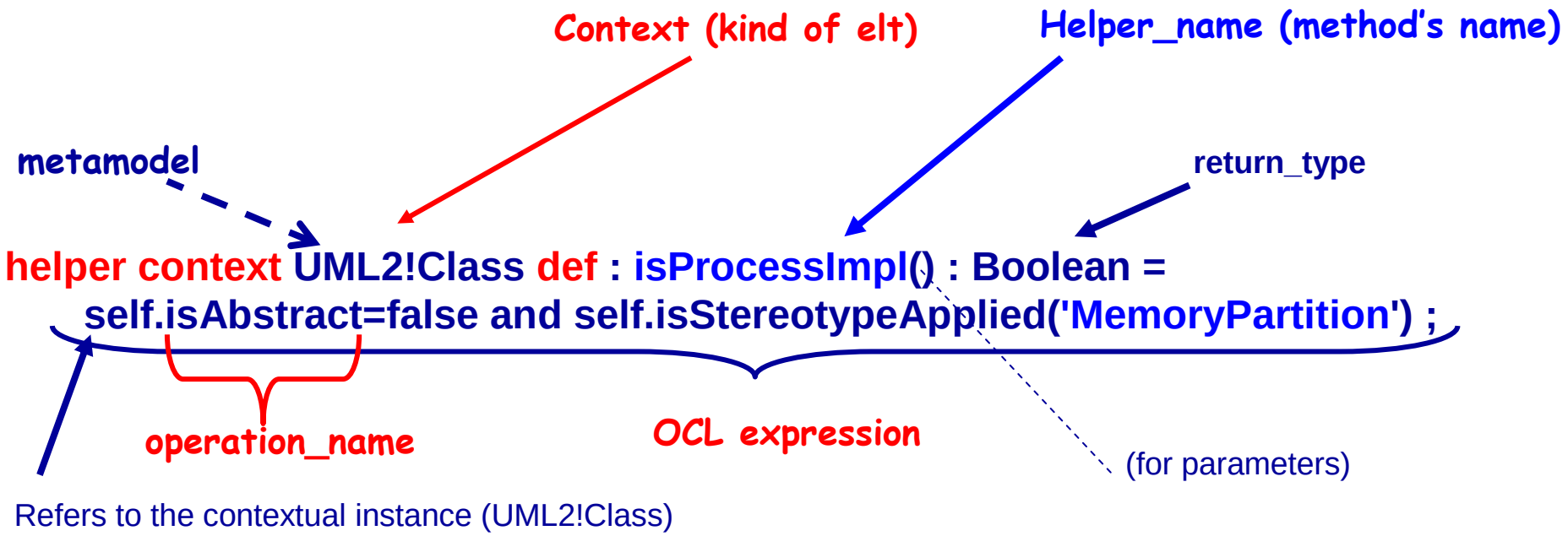


5/ Reusable ATL code: helpers

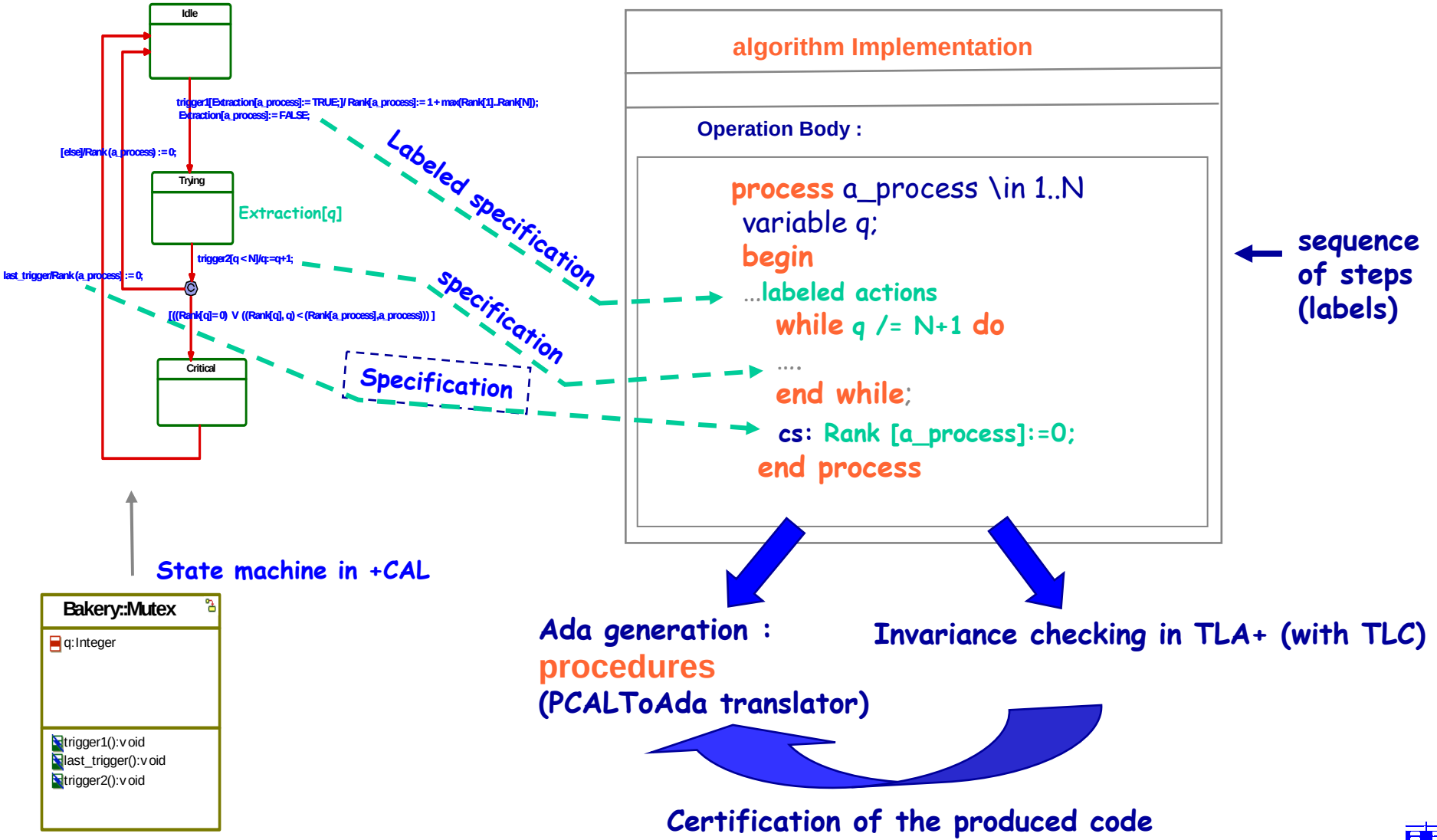
Example on : MemoryPartition → Process implementation

(if non abstract and if the UML stereotype is MemoryPartition)

ATL helpers: (java) methods within ≠ kinds of ATL units



6/ Automated Tool Support for c-limb activities: certifying the generated code



- ANTLR v3.1.2 ➔ Ada 95 Code generation from the +CAL (p)
  - ANTLR grammar, generates lexer, parser, AST, rewrite mapping rules to the grammar

- **2 simple commands** java pcal2Ada.trans pgm translate +CAL into Ada95 / java pcal.trans Algorithm → +CAL into TLA+

- The invariant to check is **no 2 process are in state cs**
- IsMutex is a TLA+ operator defined just after the algorithm:

isMutex == ∀ i ,k ∈ 1 .. N : (i /= k)) : => ¬((pc[i] = "cs") ∧(pc[k] = "cs"))

  - Then : we put **assert answer = isMutex** to check the algorithm with TLC
  - we simulate the algorithm (run)

8/ Specifying a synchronization algorithm in +CAL

other processes know if some number requests are in progress or not

number of the request

```
--algorithm bakery
variables Extraction = [k ∈ 1..N |-> FALSE],
Rank= [m ∈ 1..N|-> 0];
process a_process ∈ 1..N
  variable q;
  begin
  Extraction[a_process]:= TRUE;
  Rank[a_process]:= 1 + max(Rank[1]..Rank[N]);
  Extraction[a_process]:= FALSE;
  q:=1;
  while q ≠ N+1 do
    while (Extraction[q])
    do skip;
    end while;
    while ((Rank[q]≠ 0) ∧ ((Rank[q], q) < (Rank[a_process],a_process)))
     do skip;
    end while;
  q:=q+1;
  end while;

  \*The critical section
  Rank [a_process]:=0;
  \* non-critical section...
end process
end algorithm
```

labeled actions

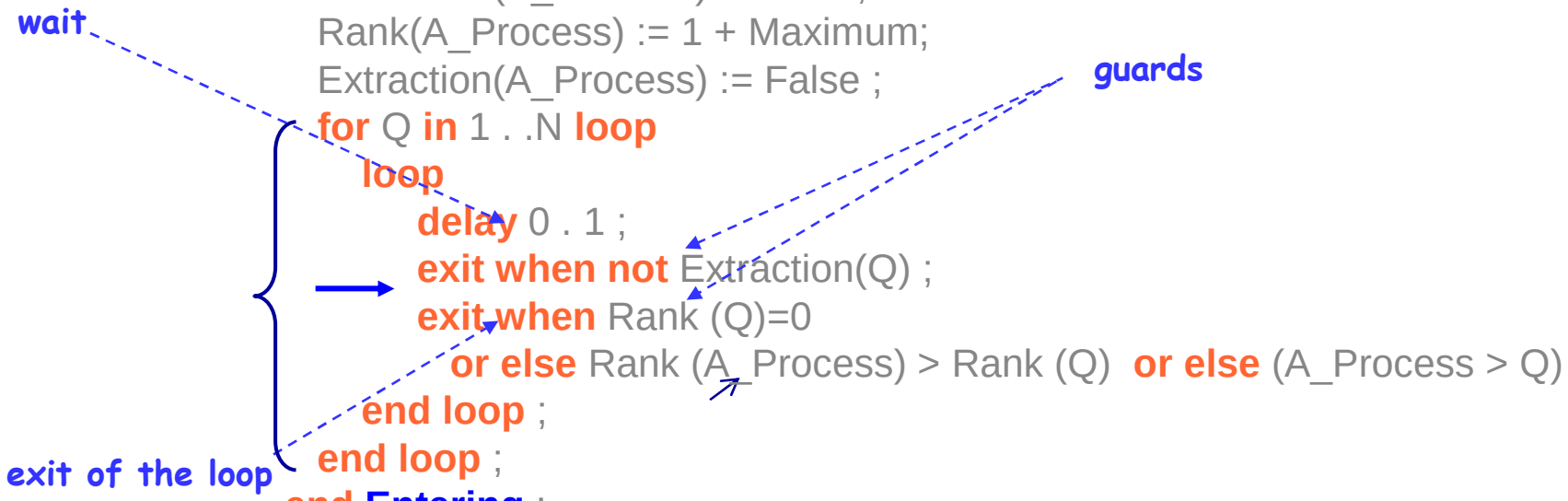
The "for" loop does not exist

attribution of the ticket … " busy waiting" (trying)

wait until other processes with higher priorities have finished their job

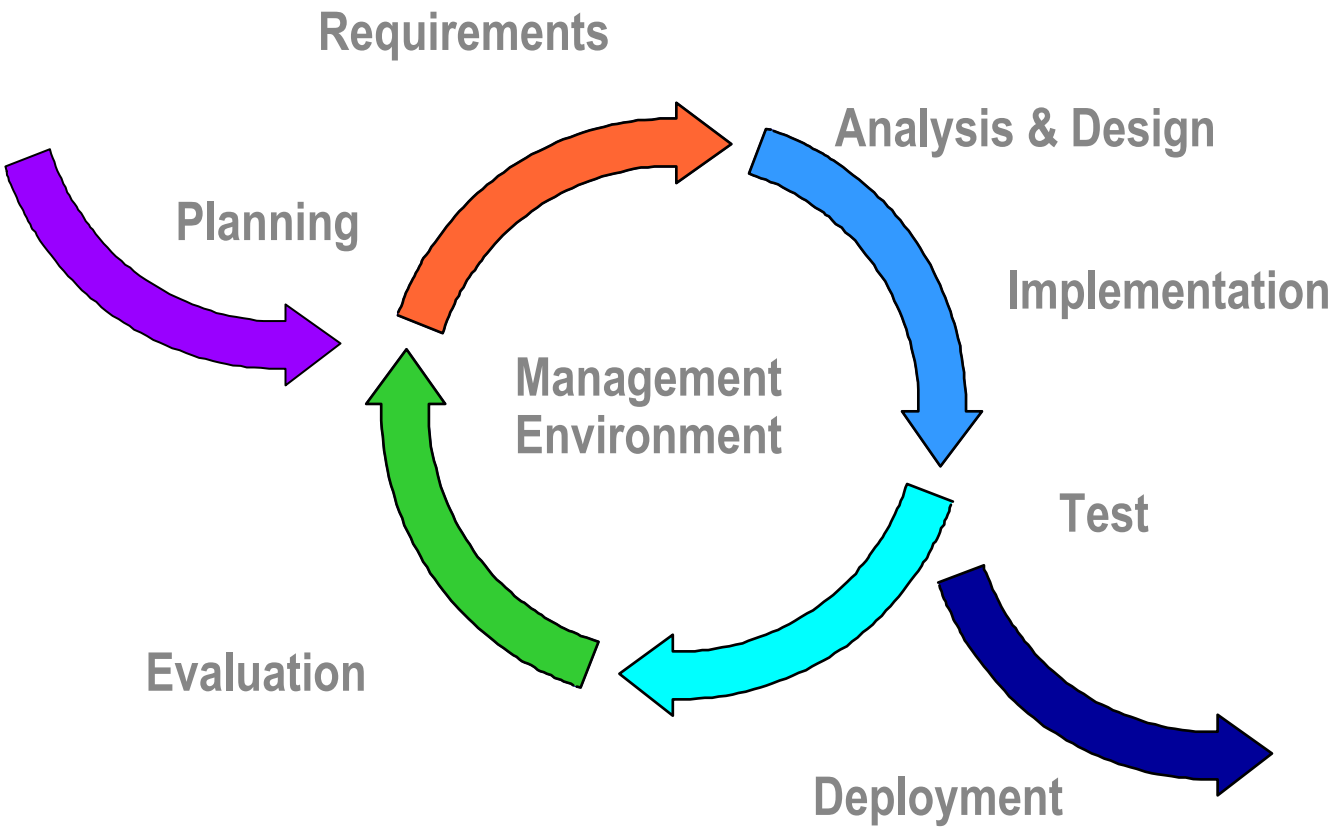
check on pid process

9/ PCAL2Ada →Ada 95 pgm



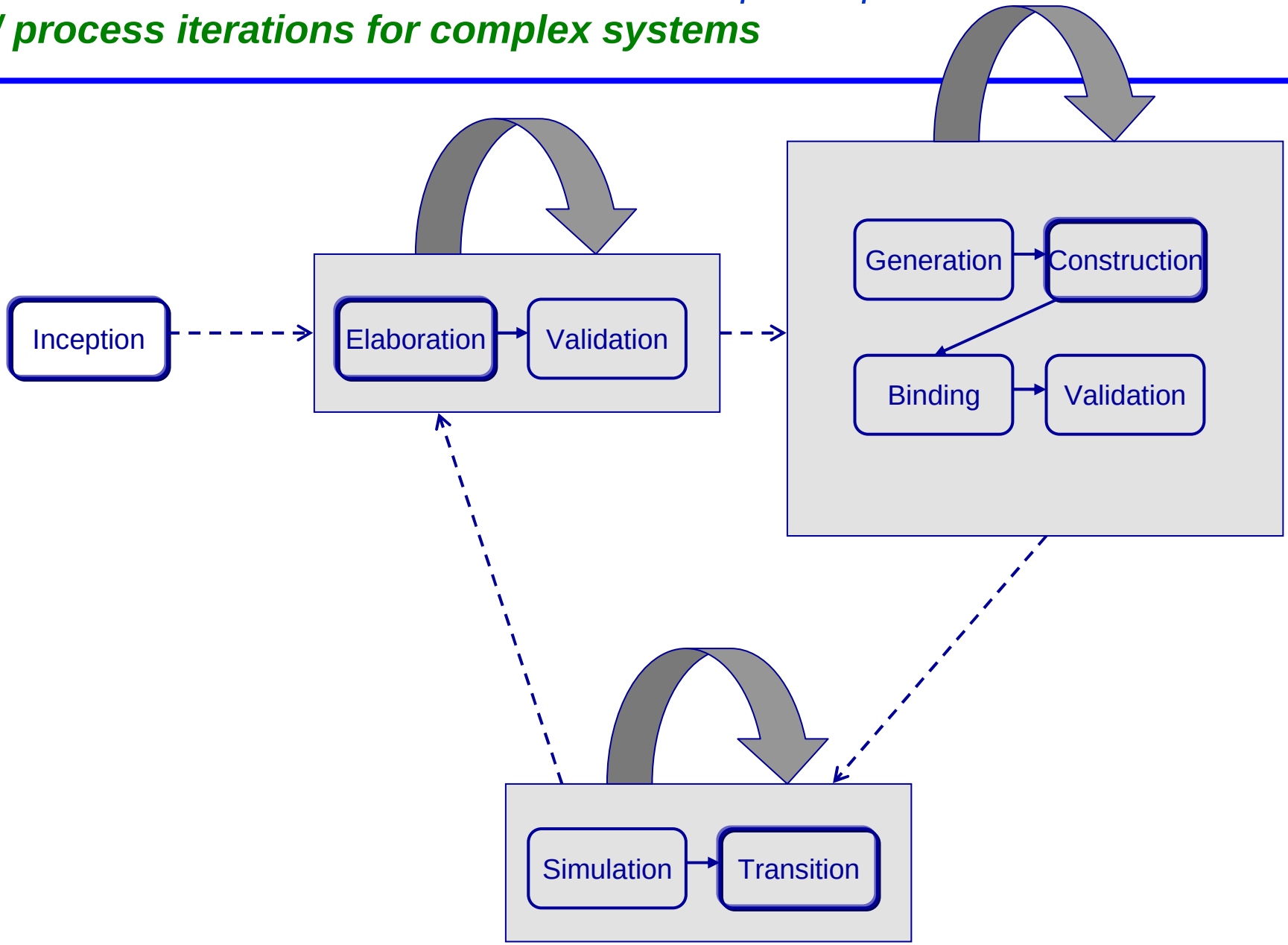

```
procedure Entering (A_Process : in Proc_Index) is
begin
   Extraction(A_Process) := True ;
   Rank(A_Process) := 1 + Maximum;
   Extraction(A_Process) := False ;
   for Q in 1 . .N loop
      loop
         delay 0 . 1 ;
         exit when not Extraction(Q) ;
         exit when Rank (Q)=0
            or else Rank (A_Process) > Rank (Q)  or else (A_Process > Q)
      end loop ;
   end loop ;
end Entering ;
--
-- Exit Protocol
procedure Way_Out (A_Process : in (Proc_Index) is
begin
   Rank (A_Process) := 0;
end Way_Out ;
```

1/ Each RUP (phase) iteration results in an executable release

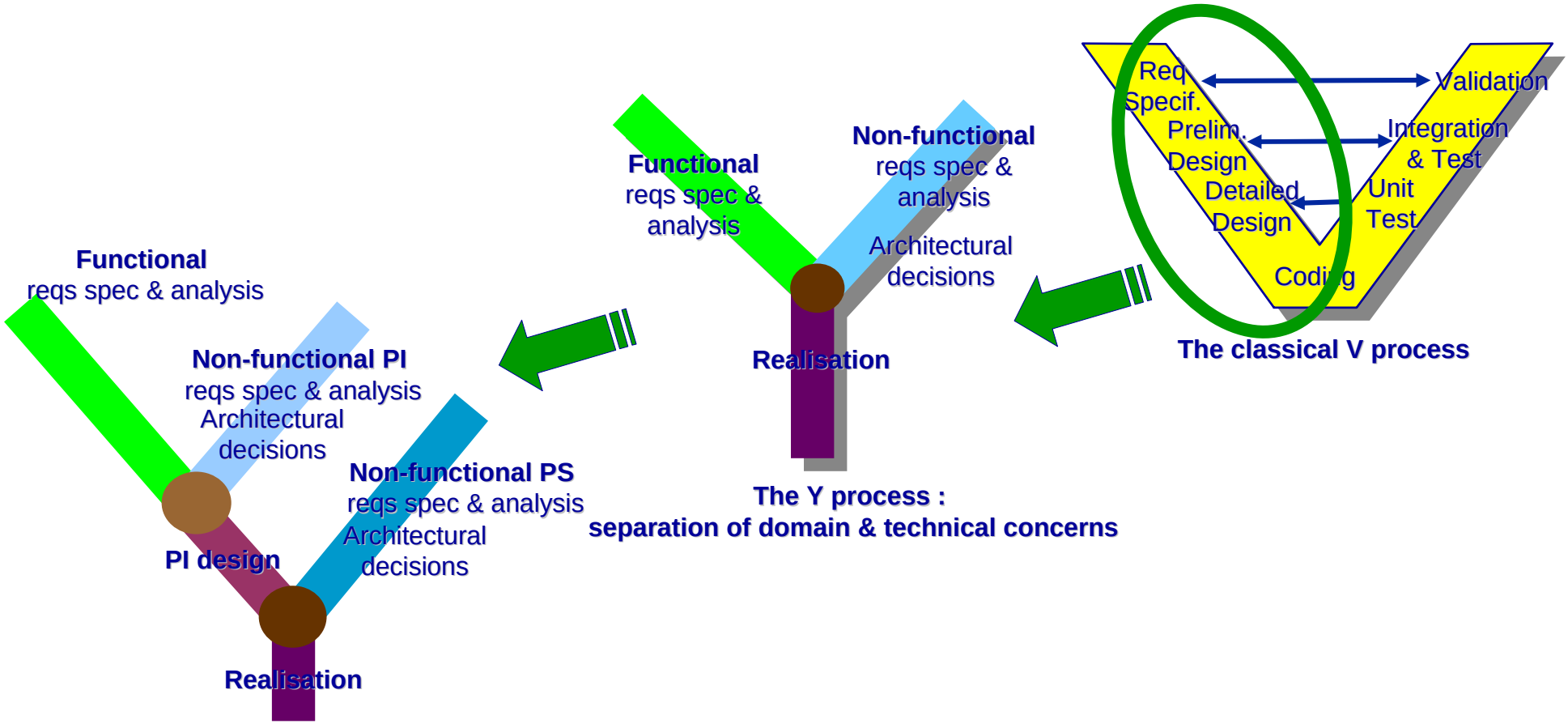


In an **iteration**, you walk through all disciplines

2/ process iterations for complex systems



3/ MDE Life-cycle evolution

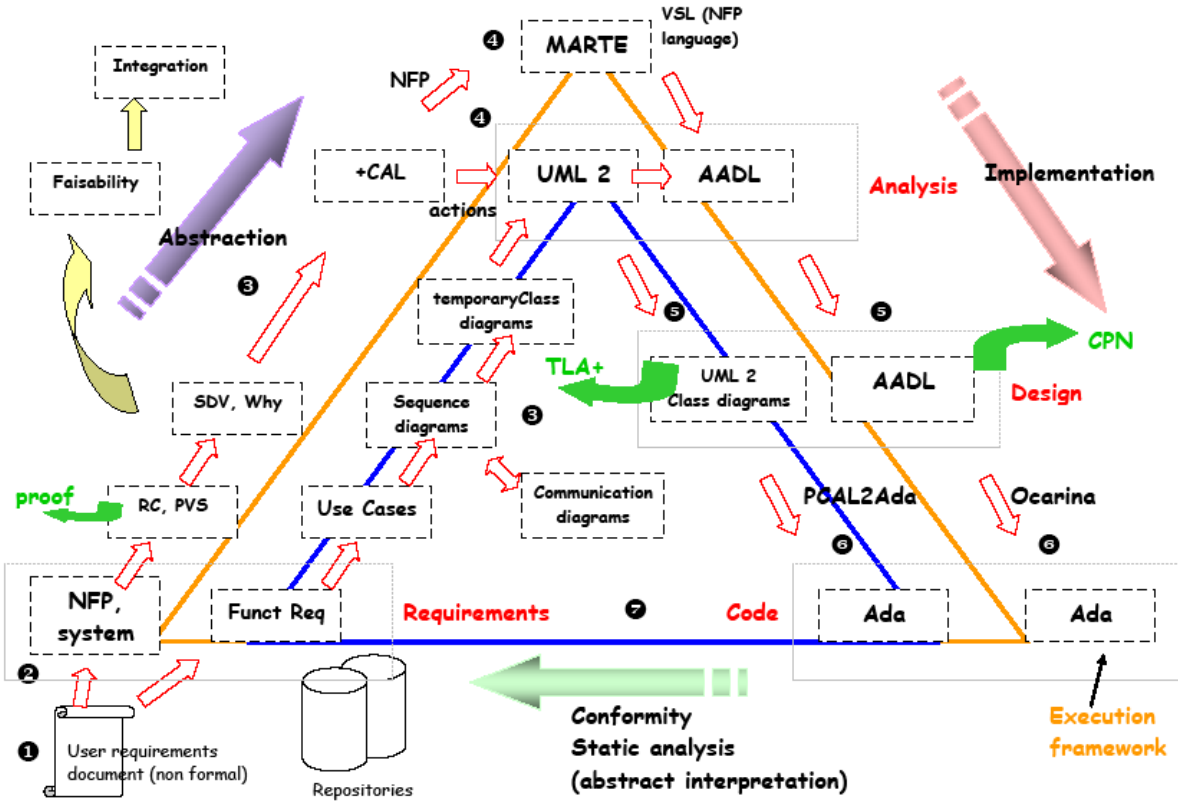


4/ C-Method and its lifecycle guided by the abstraction levels

Non Functional

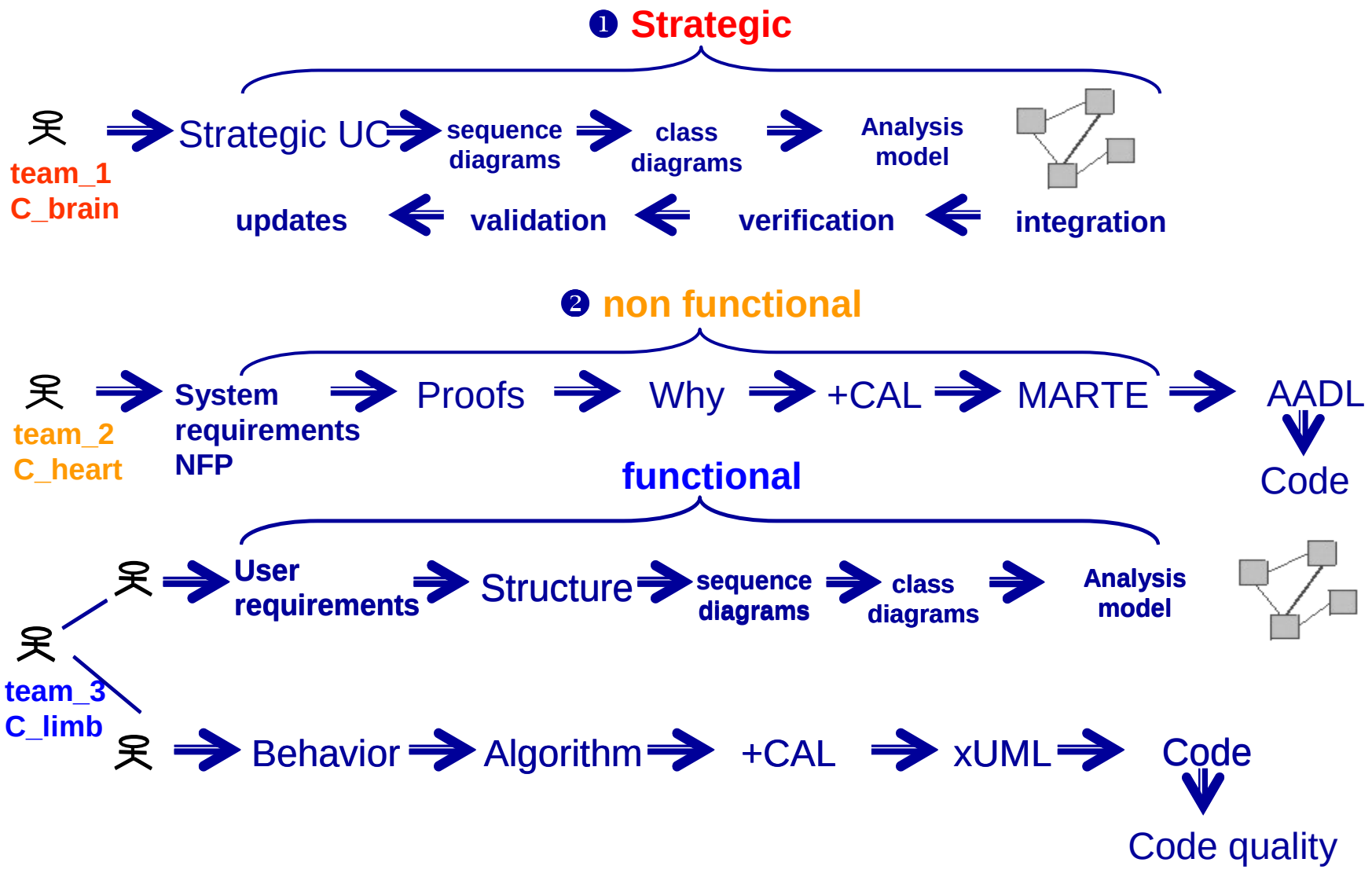
Functional

Proofs / Verification



- The strategy: multiple and // preparations of the other phases → seamless transitions
  - Activities are not sequential
  - Many strata in the requirement phases
  - 20% models generate 80% code
- 3 very large sub-processes to establish the guidelines
  - **C-brain** → software skeleton, global integration, verification
  - **C-heart** → architecture and execution framework
  - **C-limb** → functional part, final realization

6/ The C-Method software development sub-processes



- b1- manage the whole software development process
- b2- classify the requirements (strategic, user, system)
- b3- formalize the strategic requirements
- b4- check the requirements to be conform to the non-formal requirements
- b5- drive the strategic use cases
- b6- control the Analysis process
- b7- drive the integration process
- b8- drive the verification process until the last conformance step

6/ The c-heart activities

- h1- extract the non functional properties from the first requirement
- h2- formalize them
- h3- check they are conform to the non-formal requirements
- h4- translate them in an algorithm language
- h5- prove them
- h6- build the whole architecture analysis model (MARTE)
- h7- prepare the software binding
- h8- drive the MARTE2AADL transformation
- h9- trace the NFP during the model transformation into AADL
- h10- drive the scheduling analysis
- h11- optimize the architecture
- h12- bind the software on the hardware
- h13- drive the execution framework generation
- h14- drive the execution framework verification

6/ The c-limb activities

- l1- drive the functional requirements
- l2- formalize them
- l3- check their compliance to the non-formal requirements
- l4- lead the structural analysis and design
  - l41- identify the classes
  - l42- **extract the complex behavioral parts for l5**
  - l43- integrate and complete the skeletton
  - l44- generate a preliminary code skeletton
- l5- lead the behavioral analysis and design
  - l51- formalize algorithms in +CAL
  - l52- translate them into TLA+ specifications and check them with relevant invariants
  - l53- integrate the +CAL algorithms in the actions
  - l54- **lead the +CAL2Ada code generation**
- l6- test the binding
- l7- validate the integration
- l8- lead the functional validation
- l9- verify the consistency with the requirements (abstract interpretation)
- l10- manage the upgrade process until the death of the product

Future works

- A certified translator **PCAL2Ada (written in PVS)**
- Integration of +CAL in a professional modeling tool
- Automate the abstraction phase of the C-Method (IA)
- Creation of user group / working group at the OMG

Conclusions

- New phases
- New activities
- Other iteration types
- New lifecycle, new method
- Intermediate languages
- Languages integration techniques
- Same overall logic: distribution of activities along the lifecycle